



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁵ : G06F 1/00	A1	(11) International Publication Number: WO 92/20021 (43) International Publication Date: 12 November 1992 (12.11.92)
<p>(21) International Application Number: PCT/US92/03608</p> <p>(22) International Filing Date: 1 May 1992 (01.05.92)</p> <p>(30) Priority data: 697,652 8 May 1991 (08.05.91) US</p> <p>(71) Applicant: DIGITAL EQUIPMENT CORPORATION [US/US]; 146 Main Street, Maynard, MA 01754 (US).</p> <p>(72) Inventor: WYMAN, Richard, Mark ; 410 Second Avenue, South #108, Kirkland, WA 98033 (US).</p> <p>(74) Agents: NATH, B., Ram et al.; c/o Joyce D. Lange, Digital Equipment Corporation, 111 Powdermill Road, May- nard, MA 10754 (US).</p>		<p>(81) Designated States: AT, AT (European patent), AU, BB, BE (European patent), BF (OAPI patent), BG, BJ (OAPI patent), BR, CA, CF (OAPI patent), CG (OAPI patent), CH, CH (European patent), CI (OAPI patent), CM (OAPI patent), DE, DE (European patent), DK, DK (European patent), ES, ES (European patent), FI, FR (European patent), GA (OAPI patent), GB, GB (European patent), GN (OAPI patent), GR (European patent), HU, IT (European patent), JP, KP, KR, LK, LU, LU (European patent), MC (European patent), MG, ML (OAPI patent), MR (OAPI patent), MW, NL, NL (European patent), NO, PL, RO, RU, SD, SE, SE (European patent), SN (OAPI patent), TD (OAPI patent), TG (OAPI patent).</p> <p>Published <i>With international search report.</i> <i>With amended claims.</i></p>
<p>(54) Title: LICENSE MANAGEMENT SYSTEM</p> <p>(57) Abstract</p> <p>A distributed computer system employs a license management system to account for software product usage. A management policy having a variety of alternative styles and contexts is provided. Each licensed program upon start-up makes a call to a license server to check on whether usage is permitted, and the license server checks a database of the licenses, called product use authorizations, that it administers. If the particular use requested is permitted, a grant is returned to the requesting user node. The product use authorization is structured to define a license management policy allowing a variety of license alternatives by values called "style", "context", "duration" and "usage requirements determination method". The license administration may be delegated by the license server to a subsection of the organization, by creating another license management facility duplicating the main facility. The license server must receive a license document (a product use authorization) from an issuer of licenses, where a license document generator is provide. A mechanism is provided for one user node to make a call to use a software product located on another user node; this is referred to as a "calling card", by which a user node obtains permission to make a procedure call to use a program on another node.</p>		

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FI	Finland	MI	Mali
AU	Australia	FR	France	MN	Mongolia
BB	Barbados	GA	Gabon	MR	Mauritania
BE	Belgium	GB	United Kingdom	MW	Malawi
BF	Burkina Faso	GN	Guinea	NI	Netherlands
BG	Bulgaria	GR	Greece	NO	Norway
BJ	Benin	HU	Hungary	PL	Poland
BR	Brazil	IE	Ireland	RO	Romania
CA	Canada	IT	Italy	RU	Russian Federation
CF	Central African Republic	JP	Japan	SD	Sudan
CG	Congo	KP	Democratic People's Republic of Korea	SE	Sweden
CH	Switzerland	KR	Republic of Korea	SN	Senegal
CI	Côte d'Ivoire	LI	Liechtenstein	SU	Soviet Union
CM	Cameroon	LK	Sri Lanka	TD	Chad
CS	Czechoslovakia	LU	Luxembourg	TG	Togo
DE	Germany	MC	Monaco	US	United States of America
DK	Denmark	MG	Madagascar		
ES	Spain				

BEST AVAILABLE COPY

LICENSE MANAGEMENT SYSTEM

* * * * *

BACKGROUND OF THE INVENTION

5 This invention relates to methods of operation of computer systems, and more particularly to a method and system for managing the licensing of software executed on computer systems.

 In U.S. Patent 4,937,863, issued to Robert, Chase and Schafer and assigned to Digital Equipment Corporation, the
10 assignee of this invention, a Software Licensing Management System is disclosed in which usage of licensed software may be monitored in a computer system to determine if a use is within the scope of a license. The system maintains a database of licenses for software products, and stores a unit value
15 indicating the number of licensing units for each product.

 When a user wishes to use a licensed product, a message is sent to the central license management facility requesting a license grant. In response to this message, the facility accesses the database to see if a license exists for this product, and, if
20 so, whether units may be allocated to the user, depending upon the user's characteristics, such as the configuration of the platform (CPU) which will execute the software product. If the

license management facility determines that a license can be granted, it sends a message to the user giving permission to proceed with activation of the product. If not, the message denies permission.

5 While the concepts disclosed in the patent 4,937,863 are widely applicable, and indeed are employed in the present invention, there are additional functions and alternatives that are needed in some applications. For example, the license management system should allow for simultaneous use of a wide
10 variety of different licensing alternatives, instead of being rigidly structured to permit only one or only a few. When negotiating licenses with users, vendors should have available a wide variety of terms and conditions, even though a given vendor may decide to narrow the selection down to a small
15 number. For example, a software product may be licensed to a single individual for use on a single CPU, or to an organization for use by anyone on a network, or for use by any users at terminals in a cluster, or only for calls from another specific licensed product, or any of a large number of other
20 alternatives. A vendor may have a large number of products, some sold under one type of license and some under others, or a product may be a composite of a number of features from one or more vendors having different license policies and prices; it would be preferable to use the same license
25 management system for all such products.

Distributed computing systems present additional licensing issues. A distributed system includes a number of processor nodes tied together in a network of servers and clients. Each node is a processor which may execute programs locally, and may
5 also execute programs or features (subparts of programs) via the network. A program executing on one node may make remote procedure calls to procedures or programs on other nodes. In this case, some provision need be made for defining a license permitting a program to be executed in a distributed manner
10 rather than separately on a single CPU, short of granting a license for execution on all nodes of a network.

In a large organization such as a company or government agency having various departments and divisions, geographically dispersed, a software license policy is difficult to administer
15 and enforce, and also likely to be more costly, if individual licenses are negotiated, granted and administered by the units of the organization. A preferred arrangement would be to obtain a single license from the software producer, and then split this license into locally-administered parts by delegation. The
20 delays caused by network communication can thus be minimized, as well as budgetary constraints imposed on the divisions or departments. Aside from this issue of delegation, the license management facility may best be operated on a network, where the licensing of products run on all nodes of the network may be
25 centrally administered. A network is not necessary for use of

the features of the invention however, since the license management can be implemented on a single platform.

Software products are increasingly fragmented into specific functions, and separate distribution of the functions can be unduly expensive. For example, a spreadsheet program may have separate modules for advanced color graphics, for accessing a database, for printing or displaying an expanded list of fonts, etc.

Customers of the basic spreadsheet product may want some, none or all of these added features. Yet, it would be advantageous to distribute the entire combination as one package, then allow the customer to license the features separately, in various combinations, or under differing terms. The customer may have an entire department of the company needing to use the spreadsheet every day, but only a few people who need to use the graphics a few days a month. It is advantageous, therefore, to provide alternatives for varied licensing of parts or features of software packages, rather than a fixed policy for the whole package.

Another example of distribution of products in their entirety, but licensing in parts, would be that of delivering CD ROMs to a customer containing all of the software that is available for a system, then licensing only those parts the customer needs or wishes to pay fees for rights to use. Of course, the product need not be merely applications programs, operating systems, or traditional executable code, but instead

could also include static objects such as printer fonts, for example, or graphics images, or even music or other sound effects.

As will be explained below, calling and caller authorizations are provided in the system according to one feature of the invention, in order to provide technological support for a number of business practices and solve technical problems which require the use of what is called "transitive licensing." By "transitive licensing" is meant that the right to use one product or feature implies a right to use one or more other products or features. Transitive licenses are similar to group licenses in that both types of license consist of a single instrument providing rights of use for a plurality of products. However, transitive licenses differ from group licenses in that they restrict the granted rights by specifying that the licensed products can only be used together and by further specifying one or more permitted inter-product calling/caller relationships. Some examples may help to clarify the use and nature of a transitive license: the examples to be explained are (1) two products sold together, (2) a give-away that results from narrow choices of licensing alternatives, (3) a client licensing method in a client/server environment, (4) impact of modular design, and (5) the impact of distributed design.

A software vendor might have two products for sale: the

first a mail system, and the second a LEXISTM-like content-based text retrieval system. Each of these products might be valued at \$500 if purchased separately. Some customers would be satisfied by purchasing the rights to use only one of these products. others might find that they can justify use of both. In order to increase the likelihood that customers will, in fact, purchase both products, it would not be surprising if the software vendor offered his potential customers a volume discount, offering the two products for a combined price of \$800. The customers who took advantage of this combined offer would find that they had received two products, each of which could be exploited to its fullest capabilities independently from the other. Thus, these customers would be able to use the content based retrieval system to store and retrieve non-mail documents. However, from time to time, the vendor may discover that particularly heavy users of mail wish to be able to use the content based retrieval system only to augment the filing capabilities provided by the standard mail offering. It is likely that many of these potential customers would feel that \$800 is simply too much to pay for an extended mail capability. The vendor might then consider offering these customers a license that grants mail users the right to use the content-based retrieval system only when they are using mail and prohibits the use of content based retrieval with any other application that might be available on the customers system. This type of license is referred to below a "transitive

license," and it might sell for \$600.

Another example is a relational database product (such as that referred to as RdbTM) designed for use on a particular operating system, e.g., VMS. This relational database product has two components: (1) A user interface used in developing new databases, and (2) a "run-time" system which supports the use of previously developed databases. The developers of the database product might spend quite a bit of effort trying to get other products made by the vendor of the database product to use it as a database instead of having those other products build their own product-specific databases. Unfortunately, the other product designers may complain that the cost of a run-time license for the database product, when added to the cost of licenses for their products, would inevitably make their products uncompetitive. Thus, some mechanism would be needed that would allow one or another of the vendor's products to use the run-time system for the relational database product in a "private" manner while not giving unlicensed access to products of other vendors. No such mechanism existed, prior to this invention; thus, the vendor might be forced to sell the right to use its run-time system for the database product with its proprietary operating system license.

Clearly, this combined license would make it possible for the vendor's products to use its database product without increasing their prices; however, it also would make it possible for any

customers and third-parties to use the database product without paying additional license fees. However, had the system of the invention been available, the vendor could have granted transitive licenses for the run-time component of its database product to all the vendor's products. Essentially, these licenses would have said that the database run-time could be used without an additional license fee if and only if it was used in conjunction with some other of the vendor's products. Any customer wishing to build a new relational database application or use a third-party application that relied on the vendor's database product would have had to pay the vendor for its database run-time license.

A proposed client/server licensing method provides yet another example of a problem which could be solved by transitive licensing. Typically, a client is only used by one user at a time, while a server can support an arbitrary number of clients depending on the level of client activity and the capacity of the machine which is supporting the server. While traditionally, server/client applications have been licensed according to the number of clients that a server could potentially support, this may not be the most appropriate method for licensing when the alternatives afforded by the invention are considered. The business model for the proposed client/server method requires that each client be individually licensed and no explicit licensing of servers is required to

support properly licensed clients.

Such a licensing scheme makes it possible to charge customers only for the specific number of clients they purchase. Additionally, it means that a single client can make use of more than one server without increasing the total cost of the system.

The solution to this transitive licensing problem would be to provide a mechanism that would allow the clients to obtain license unit allocations and then pass a "proof" of that allocation to any servers they may wish to use. Servers would then support any clients whose proofs could be verified to be valid. On the other hand, if a client that had not received a proof of allocation attempted to use a server, the server would obtain a license allocation for that client session prior to performing any services. Such a solution has not been heretofore available.

As the complexity and size of the software systems provided to customers increases, it is found that the actual solution provided to customers is no longer a single product. Rather, customers are more often now offered solutions which are built up by integrating an increasing number of components or products, each of which can often stand alone or can be part of a large number of other solutions. In fact, a product strategy may rely almost exclusively on the vendor's engineering and selling a broad range of specialized components that can only be

fully exploited when combined together with other components into a larger system. Such components include the relational data base runtime system mentioned above, mail transport mechanisms, hyperinformation databases, document format conversion services, time services, etc. Because these components are not sold on their own merits, but rather on their ability to contribute to some larger system, it is unlikely that any one customer will be receiving the full abstract economic value of any one of the components once integrated into a system. Similarly, it can be observed that the value of any component once integrated into a larger system varies greatly from system to system. Thus, it may be found that a mail transport mechanism contributes a large part of a system whose primary focus is mail, however, it will contribute proportionally less of the value of a system that provides a broader office automation capability. As a result of these observations, the job of the business analyst who is attempting to find the "correct" market price for each component standing on its own, is more complex. In reality, the price or value of the component can only be determined when considering the contribution of that component to the full system or solution in which it is integrated. Attempting to sell the components at prices based on their abstract, independent values will simply result in overpriced systems.

Transitive license styles are particularly suited to

dealing with pricing of modular components, since component prices can be clearly defined in relation to the other components or systems which they support. Thus, a vendor can charge a price of \$100 for the right to use a mail transport system in conjunction with one product, yet charge \$200 for the use of the same mail transport system when used by another product.

In addition to the "business" reasons for wanting to support transitive licensing, there is also a very good technical reason that arises from the growing tendency of developers to build "distributed products" as well as the drive toward application designs that exploit either tightly or loosely coupled multiprocessor systems; the availability and growing use of remote procedure calls has contributed to this tendency. This technical problem can be seen to arise when considering a product which has a number of components, each of which may run in a different process space and potentially on a different computer system. Thus, there might be a mail system whose user interface runs on one machine, its "file cabinet" is supported by a second machine and its mail transport system runs on yet a third machine. The simple question which arises is: "Which of the three components should check for licenses?" Clearly it must be ensured that no single component can be used if a valid license is not present. Thus, the answer to the question will probably be that all three components should check

for licenses.

However, the question is then presented: "Where are the licenses to be located?".

This can become more complex.

5 Increasingly, the distributed systems being built are being
designed so that it is difficult to predict on which precise
machine any particular component will run. Ideally, networks
are supposed to optimize the placement of functions
automatically so that the machine with the most available
10 resource is always the one that services any particular request.
This dynamic method of configuring the distribution of function
servers on the network makes it very difficult for a system
or network manager to predict which machines will run any
particular function and thus very difficult for him to decide on
15 which machines software licenses should be loaded.

Even if a system manager could predict which machines would
be running the various application components and thus where the
license units should be loaded, the situation would still be
less than ideal. The problem arises from the fact that each of
20 the components of the application would be independently
making requests for license unit allocations. This behavior
will result in a difficult problem for anyone trying to decide
how many license units are required to support any one product.
Given the mail example, the problem wouldn't exist if it were

assumed that all three components (i.e., user interface, file cabinet, and transport system) were required by the design of the mail system to be in use simultaneously. If this were the case, it could be simply assumed that supporting

5 a single activation of the mail system would require three units. However, in areal mail system, it will be inevitably discovered that many users will only be using just the user-interface and file-cabinet components of the system at one time.

10 Thus, there will be some unused units available which could be used to authorize additional users. This situation might not be what is desired by the software vendor.

The problem of providing license support to multi-component products which are dynamically configured could be solved by
15 viewing each of the product components as a distinct licensable product and by treating the problem as one of transitive licensing, but a mechanism for accomplishing this has not been available. Essentially, a single license document would be created that stated that if any one of the components had
20 successfully obtained a license to run, it could use this grant to give it the right to exploit the other components. Thus, in the example above, the user might start the mail system by invoking its user interface.

This user interface code would then query the license management
25 facility for a license allocation and once it has received that

allocation, it would pass a proof of allocation to the other mail components that it uses. Each of the other components would request that the license management system validate that the "proof" is valid prior to performing any service; however, 5 none of the other components would actually require specific allocations to be made to them. In this way, the complexity of licensing and managing networks of distributed applications can be significantly reduced.

SUMMARY OF THE INVENTION

10 The invention its broad form resides in a method of managing use of licensed software items, comprising the steps of:

maintaining a store of license authorizations for said software items; each license authorization including an indication of license management policy for a software item 15 having plurality of sets of policy components, said policy components in each set providing alternatives, to produce a composite policy simultaneously using a policy component from each of said sets; sending a request by a user of one of said software items to obtain permission to use said software item; 20 said request identifying the user and said software item; accessing said store to obtain information from said license authorization for said software item, in response to said

request, and comparing said identification of said user and said software item with said information, and with constraints imposed by said plurality of sets of license management policy components, to produce a grant or refusal of said request;
5 sending said grant or refusal to said user.

In accordance with one embodiment of the invention, a license management system is used to account for software product usage in a computer system. The system employs a license management method which establishes a management policy having a variety of
10 simultaneously-available alternative styles and contexts. A license server administers the license, and each licensed product upon start-up makes a call to the license server to check on whether usage is permitted, in a manner similar to that of patent 4,937,863. The license server maintains a store of the
15 licenses, called product use authorizations, that it administers. Upon receiving a call from a user, the license server checks the product use authorization to determine if the particular use requested is permitted, and, if so, returns a grant to the requesting user node. The license server maintains
20 a database of product use authorizations for the licensed products, and accesses this database for updating and when a request is received from a user. While this license management system is perhaps of most utility on
a distributed computer system using a local area network, it is
25 also operable in a stand-alone or cluster type of system. In a

distributed system, a license server executes on a server node and the products for which licenses are administered are on client nodes. However, the license management functions and the licensed products may be executing on the same processor in some
5 embodiments.

 The product use authorization is structured to define a license management policy allowing a variety of license alternatives by components called "style", "context", "duration" and "usage requirements determination method". The style
10 may be allocative or consumptive. An allocative style means the units of the license may be allocated temporarily to a user when a request is received, then returned to the pool when the user is finished, so the units may be reused when another user makes a request. A consumptive style means the units are deducted
15 from an available pool when a user node makes a valid request, and "consumed", not to be returned for reuse. The context value defines the context in which the use is to be allowed, such as on a particular network, by a particular type of CPU, by a particular user name, by a particular process, etc. The
20 duration value (used in conjunction with the style component) concerns the time when the license units are to be deducted from the available pool of units, whether at the time of request, after a use is completed, etc. A usage requirements determination method may be specified to define or
25 provide information concerning the number of license units

charged in response to a license request from a user node; for example, some CPU platforms may be charged a larger number of license units than others. A table may be maintained of usage requirements, and the determination method may specify how to
5 access the table, for example. The important point is that the user node (thus the software product) can only make a request, identifying itself by user, platform, process, etc., and the license management facility calculates whether or not the license can be granted (that is, units are available for
10 allocation), without the user node having access to any of the license data or calculation. There is a central facility, the license server, storing the license documents, and, upon request, telling the licensed products whether they can operate under the license terms.

15 An important feature of one embodiment is that the license administration may be delegated to a subsection of the organization, by creating another license management facility duplicating the main facility. For example, some of the units granted in the product use authorization may be delegated to
20 another server, where the user nodes serviced by this server make requests and receive grants.

The license management facility cannot create a license itself, but instead must receive a license document (a product use authorization) from an issuer of licenses. As part of the

overall license management system of the invention, a license document generator is provided which creates the product use authorizations under authority of the owner of the software, as negotiated with customers. Thus, there are three distinct rights in the overall license management facility of the invention: (1) the right to issue licenses, (2) the right to manage licenses, and (3) the right to use the licensed products. Each one of these uses the license document only in prescribed ways. The license issuer can generate a license document. The license manager (or license server as referred to herein) can grant products the right to use under the license, and can delegate parts of the licensed units for management by another server, as defined by the license document; the way of granting rights to products is by responding to certain defined calls from the products. And, the licensed products can make certain calls to the license server to obtain grants of rights based upon the license document, inquire, or report, but ordinarily cannot access the document itself.

As explained above, transitive licensing is an important feature of one embodiment. This is the provision of a mechanism for one user node to get permission to use another software product located on another user node; this is referred to as a calling authorization and a caller authorization, using a "calling card," and these are examples of the optional features which must be specifically permitted by the product use

authorization. A user node must obtain permission to make a procedure call to use a program on another node; this permission is obtained by a request to the license server as before, and the permission takes the form of a calling card. When a calling card is received by a second node (i.e., when the procedure call is made), a request is made by the second node to the license server to verify (via the product use authorization) that the calling card is valid, and a grant sent to the user node if allowed. In this manner, all nodes may have use of a program by remote calls, but only one consumes license units.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as other features and advantages thereof, will be best understood by reference to the detailed description of specific embodiments which follows, when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a diagram in block form of a distributed computer system which may be used to implement the license management operations according to one embodiment of the invention;

Figure 2 is a diagram of the content of a license document

or "product use authorization" generated by the license document generator and stored by the license server in the system of Figure 1;

Figure 3 is a diagram of the alternatives for license style, context and duration making up the license management policy implemented in the system of Figure 1, according to one embodiment of the invention;

Figure 4 is a diagram of an example of a fragment of a license use requirements table (LURT) used in the system of Figure 1, according to one embodiment of the invention;

Figure 5 is a logic flow chart of a program executed by a user node (client), in the system of Figure 1, according to one embodiment of the invention;

Figure 6 is a logic flow chart of a program executed by a license server, in the system of Figure 1, according to one embodiment of the invention; and Figure 7 is a diagram of the calls and returns made in an example of use of calling cards in the system of Figure 1.

DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS

Referring to Figure 1, a license management facility according to one exemplary embodiment of the invention is centered around a license server 10, which typically includes a CPU located in the customer's main office and executing a license management program 11 as will be described, under an operating system 12. The license server 10 communicates with a number of delegates 13 which likewise include CPUs in departments or divisions of the company or organization, each also executing a license management program 14 under an operating system 15. The license management program 14 is the same as the program 11 executing on the main server 10; the only difference in the functions of server 10 and servers 13 is that the latter have a delegated subset of the license units granted to the server 10, as will be described. The CPUs 13 are in turn servers for a number of users 16, which are CPU nodes where the licensed programs 17 are actually executed. The programs 17 executing on the user CPUs 16 are applications programs (or operating systems, etc.) which have added to them units 18 and 19, according to the invention, allowing them to make inquiry to the their server 13 (or 10) before executing and to report back after executing, using a client stub 19 in the manner of remote procedure calls, in one embodiment. A user node 16 may have many different programs 17 that may be executed, and the various user nodes 16 would usually each have a set of programs 17 different from the other user nodes, all of which would be administered by the license management program 14 or 11. The

terms "program" and "licensed product" are used in reference to the element 17, but it is understood that the products being administered may be segments of programs, or functions or features called by another program, or even merely data (such as printer fonts), as well as complete stand-alone applications programs. The license server 10 communicates with the delegatee servers 13 by a network 21, as is usual in large organizations, and the delegatee servers 13 each communicate with their user nodes 16 by networks 22; these networks may be of the Ethernet, token ring, FDDI types or the like, or alternatively, the user nodes 16 may be merely a cluster of terminals on a multiuser system with the delegatee being a host CPU. The particular hardware construction of the user nodes, server nodes, communication networks, etc., and the operating systems 12 or 15, are of no concern regarding the utility of the features of the invention, the only important point being that the user CPUs 16 of the software products 17 in question can communicate readily and quickly with their respective server nodes 13 or 10. In one embodiment, remote procedure calls (RPCs) are used as the communication medium for the interfaces between components of the system, handling the inquiries and grants as will be described.

A remote procedure call is similar to a local procedure call but is made to a procedure located on a remote node, by way of a communications network.

The function of the unit 19 is that of a client stub, in a remote procedure call sense. The calls to the license server 10 are made through this stub 19, and returns are received by the stub 19 and passed on to the program 17. The stub 19 is responsible for obtaining the network addresses of other nodes on the network, such as the server 10. Also, the stub 19 is responsible for determining the context (as defined below) for passing on to the server 10. The unit 18 functions to execute a "private" type of license availability determination if this is used, rather than this task being done by the application program 17, but if the ordinary method of determination is employed (using the license server) as is usually the case, the unit 18 is merely code that starts the execution and passes calls and returns back and forth between the program 17 and the unit 19.

The license server 10, using the license management program 11, maintains a license data file 23 comprising a number of license documents or licenses (product use authorizations), and also maintains a log 24 which is a record of the usage activity of all of the user CPUs 16 of each of the licensed programs. The delegatee servers 13 would maintain similar license databases and logs. The license server 10 has no authority to originate a license, but instead must receive a license from a license issuer 25. The issuer 25 is again a CPU executing a license document generator program 26 under an operating system

27. The license issuer 25 may be under control of the producer
28 of the programs or software products being licensed, or may
be controlled by a distributor who has received the authority to
grant licenses from the producer or owner 28. The
5 communications link 30 between the license issuer 25 and the
license server 10 for delivering the license document may be in
the form of a network, or may be a phone line using modems, or
may include physical delivery by disks or CD ROMs, for example.
Likewise, the method of delivery of the software products being
10 licensed, i.e., the applications programs 17 to be executed on
the CPUs 16, is not material to the license management facility
of the invention; the products are delivered by some appropriate
means, e.g., the communications link 30 and the networks 21 and
22, by CD ROMs or disks physically distributed, etc.

15 Although shown in Figure 1 as operating on a distributed
system, in the simplest case the license management facility of
the invention may be operated on a single CPU. The license
management program 11 and the applications program 17 may be
executing on the same CPU, in which case the license document
20 would be stored in a database 23 as before, on this CPU, and the
calls from the unit 18 to the license server would be local
instead of RPCs. As in the distributed system, however, the
licensed product would still not have access to the license
document, but instead could only make inquiries to the server
25 program, even if all are executing on the same CPU.

In operation of the distributed system of Figure 1, the producer 28 gives the issuer 25 authority to grant licenses on its behalf (the producer and issuer can be a single entity or multiple entities). The license document generator program 26, under control of a user (a person), generates a license (usually the result of negotiation between the user of program 26 and a user of the server 10). This license is called a product use authorization, and it is transmitted by the link 30 to the server 10. The license management program in the server 10 stores the product use authorization in the database 23, and, if delegation is an authorized option, may distribute parts of the authorized use to the delegatee servers 13, where it is likewise stored in a database. Thereafter, administration of the license is only in response to inquiries from user nodes 16. When execution of a program 17 begins, the unit 18 is invoked to check on the availability of a license for this particular node. The unit 18 sends (as by an RPC) a request to the license management program 14 (or 11 if there is no delegatee), where the product use authorization stored in database 23 is checked to see if use is authorized. If so, a return is sent to the user node 16, granting permission to continue. When the program 17 has finished executing, the unit 18 again is invoked to signal to the license management program, again by an RPC, that the authorization is released, so the license management program can take appropriate action, e.g., log the use in log 24, etc.

To implement these operations, the license management program 11 or 14 contains several functions, including a client interface 31, a database interface 32, a management interface 33, and an interserver interface 34 for communicating
5 with the delegates 13 (if any). The client interface 31, as described below, handles the requests received from the user nodes 16, and returns resulting from these requests. The database interface 32 handles the storing and retrieval of
10 license information in the database 23, and logging license usage activity to log 24, and retrieval of this data. The management interface 33 handles the tasks of receiving the product use authorizations from the issuer 25 and maintaining the database 23 via the database interface 32. The interserver
15 interface 34 handles the task of communicating with the delegatee servers 13, including transmitting the assigned parts of the product use authorizations, or communicating with other license servers that may be separately executing the license management function; for example, calls for validating calling
20 cards may be made to another such server. If there are no delegates or no other license servers, then of course the interserver interface 34 has no function, and is idle.

The license document or "product use authorization" forming the basis for the license management activity of the program 11 on the server 10 may be illustrated as a data structure
25 containing the information set forth in Figure 2; in actual practice the product use authorization is preferably a

more abstract data arrangement, not in such a rigidly structured format as illustrated. For example, the product use authorization as well as similar documents stored in the database 23, or passed between components of the system of Figure 1, may be of the so-called tag-length-value data format, where the data structure begins with an identifying tag (e.g., PUA or product use authorization) followed by a field giving the length, followed by the value itself (the content). One type of data treatment using this tag-length-value format is an international standard referred to as ASN.1 or Abstract Syntax Notation. In any event, the document 35 illustrated in Figure 2 is merely for discussing the various items of data, rather than representing the way the information is stored. Some of the fields shown here exist at some times and not others, and some are optional; the product use authorization may also include additional fields not shown or discussed here. Also it should be noted that copies of parts of this type of document are made for the delegates, so this representation of Figure 2 is a composite of several documents used in the system of Figure 1. The document 35 includes fields 36 identifying the software product by product name, producer, version numbers, release date, etc. The issuer 25 is identified in field 37, and the licensee (usually the owner of the license server 10) identified in field 38. The essential terms of the license grant are then defined in fields 40-46. The start date and end date are specified in fields 40; these store the exact time

(date, hour, minute, second, etc.) when the license becomes valid and when it ends, so licenses may be granted to start at some future time and to end at a particular time. Note that the previous practice has been to specify only the ending date, rather than also a start date as employed here. Each of the nodes, including issuer 25, servers 10 and 13, and user nodes 16, maintain a time value by a local clock referenced to a standard, so inherent in the license management facility is the maintaining of a time standard to compare with the start and end date information in the fields 40. The units granted are specified in field 41; the units are an arbitrary quantitative measure of program usage. In a delegatee server 13, the units field 41 will have some subset of the units field in the original product use authorization. As units are granted to users 16 or delegated, the remaining units available for grant are indicated in a subfield 42 in the copy of the document used by the server. The management policy occupies fields 43-46, and includes style, context, duration and LURDM (license use requirements determination method), as will be explained. The style field 43 specifies whether the licensed units are controlled by an "allocative" style or "consumptive" style, or some other "private" algorithm, where styles are ways used to account for the consumption or allocation of the units. The context field 44 specifies the location and environment in which product use or license management occurs, i.e., a CPU or an individual user or a network, etc. Duration field 45 indicates

whether the license granted to a user is by assignment, by transaction, or immediate. The LURDM field 46 indicates the license use requirements determination method, in some cases using a license use requirements table (LURT) seen as field 47,
5 as will be described.

Additional fields 48-54 in the product use authorization 35 of Figure 2 define features such as delegation authorization, calling authorization, overdraft authorization, combination authorization, token, signature, checksum, etc. These
10 will be described in the following paragraphs.

If the delegation field 48 is true, a license server 10 may distribute license units to multiple servers 13. A time limit may be imposed, i.e., units can be delegated to other hardware systems until they time out. Delegation allows an administrator
15 to distribute units to improve response time and increase the resilience of the system. For example, the communication network 21 may include a satellite link to a remote facility where the local server 13 has a number of clients or users 16, in which case the calls to the server 13 would be completed much
20 quicker than would be the case if calls had to be made to the server 10. Also, delegation may be used as a method of allocating licensed units within a budget for administrative purposes. Usually the delegation authorization is a feature that is priced by the issuer, i.e., a license granting 1000

units with delegation authorization is priced higher than without this authorization.

The field 49 contains a calling authorization and/or a caller authorization.

5 If the caller authorization in field 49 is true, the product is permitted to receive calls from other named products requesting use of the product, and if conditions are met (identified caller is authorized) the server can grant a calling card, as described below. If the calling authorization is true, the product can
10 make calls to other products. If neither is true, then the product can neither make or receive calls using the calling card feature. Referring to Figure 1, if product 17a wishes to make a remote procedure call to a feature of product 17b running on a different user node 16, it makes a call to its server 13
15 including a request for a calling card, and, if permitted, the return to product 17a includes a calling card 49a. The product 17a then makes a call to product 17b in the usual manner of RPCs, sending along the calling card 49a, which the product 17b then verifies by a call to its server 13 before executing the
20 called procedure and issuing its return to product 17a. The feature of calling cards is important for distributed applications. For example, if a product is able to execute faster in a distributed system by assigning tasks to other CPUs, then the issue is presented of which license policy is needed,
25 i.e., does every node executing a part of the task have to be

licensed and consume or receive allocation of a unit, or just the one managing the task? This is resolved for most applications by use of this calling card concept.

The product use authorization for such a product has the calling
5 authorization field 49 enabled, so calling cards can be issued.
This feature is typically separately priced.

The combination authorization field 50 of Figure 2 determines whether or not license requests from a user node 16 can be satisfied by combining units from multiple product use
10 authorizations. It may be advantageous to purchase licenses with different policy values, and use units from certain product use authorizations only for overflow or the like. Or, for other reasons, it may be advantageous to "borrow" and "lend" units among delegated servers or user nodes. This function is
15 permitted or denied by the content of field 50.

The overdraft field 51 determines whether or not a requested allocation from a user node 16 will be nevertheless granted, even though the units available field 42 is zero or too small to permit the requested use. Overdrafts can be unlimited,
20 or a specific overdraft pool can be set up by a server 10, for a customer's internal administrative purposes. That is, the overdraft value may be unlimited in the original license, but limited or zero for internally distributed copies of the license. Thus, the product use authorization sent by the issuer

25 to the customer may have overdrafts permitted by the field 51, but the customer may deny overdraft permission for its own budgeting purposes. In any event, if overdraft is permitted, additional fees have to be paid to the issuer at some accounting period, when the logged usage from log 24 indicates the available units have been exceeded. If overdraft is denied, then the units 18 of the user nodes making request allocations are structured to inform the products 17 that a license grant is not available. The intent is not to prevent the application program from running; the license server merely informs the application whether or not the license manager determines that it is authorized to run. The application can itself be structured to shut itself down if not authorized to run, or it can be structured to shut down certain functions (e.g., ability to save files, ability to print, etc.), or it can be structured to continue in a fully functional manner. The purpose of the license management facility is not that of enforcement, nor that of "copy protection", but instead is merely that of license management.

20 An optional token field 52 is available in the product use authorization 35 of Figure 2. This field can contain comments or other information desired by the issuer or user. For example, a telephone support number may be included in the token field, then when the product 17 shows its "help screen" the number is inserted. This number would be part of the argument, i.e., data transmitted to the user node 16, when the

server 10 makes a return following a request allocation message from the user. This field may also be used to store information used in a "private" style, where the information from this field returned to the user node is employed by the application program 5 17 or the stub 19 to determine if the application can be activated.

The signature field 53 in the product use authorization 35 is a part of a validation mechanism which provides important features. This field contains a digital signature encoded to 10 reflect the data in the license itself, as well as other encoding methods not known to customers, so it cannot be duplicated unless the encoding algorithm is known. In a preferred embodiment, a so-called "public/private key" system of encoding is used for the signature field 53. The encoding 15 algorithm used to generate the signature 53 is known to the issuer 25, using a private key, and anyone knowing the public key can decode the signature to determine if it is valid but cannot determine the encoding algorithm so it cannot produce a forged signature. So, if the server 10 knows the public key 20 which is unique to the issuer 25, it can determine if a license document 35 is genuine, but it cannot itself generate license documents. However, if the server possesses a valid license document that gives it the right to delegate, then it will be assigned its own private key (different from all other issuers 25 or servers) and its delegateses 13 will be able to determine if

a valid delegated license is delivered to them as they will be given the public key for the servers 13. The field 53 will thus contain both the original signature from the issuer 25 and the license server's signature when delivered to a delegatee 13.

5 The decoding algorithm using a public key for any signatures is thus used by the license server 10 or delegatee 13 to make sure a product use authorization 35 is authentic before it is stored in the database 23. Related to the digital signature 53 is a checksum field 54, which merely encodes a value related by some

10 known algorithm to the data in the product use authorization 35 itself. This field may be used merely to check for corruption of the data as it is stored, recalled, and transmitted within the system.

That is, the checksum is used for data validation rather than

15 security.

Two concepts central to the license management system implemented using the license document or product use authorization 35 of Figure 2 are the "license units", specified in field 41 or 42 and the "context", specified in field 44.

20 License units are an abstract numerical measure of product use allowed by the license.

When a product 17 (or a function or feature of a product) makes a license-checking request, the license management program 11 on server 10 computes how many license units are required to

25 authorize this particular use of the product, and this is the license units requirement, in some cases using the LURDM field

46.

A "context" is a set of tagged values which define the location and environment in which product use or license management occurs. Context values may be specified in field 44 of the product use authorization 35 of Figure 2 to restrict the environments in which the license may be managed and in which product use may occur. A context template may also be specified in the field 44 to indicate which parts of the complete context of product use (sub-contexts) are significant in differentiating product uses for the purposes of unit allocation; when this is specified, it allows separate product uses to share license units in a controlled way.

The two general types of policies specified in field 43 are allocative and consumptive. An allocative policy grants to the holder a specific number of license units (field 41) and specifies the policy which must be used to account for the allocation of these units. A software product 17 which is being managed by an allocative license will require verification that the appropriate number of license units have been allocated to it prior to performing services to the user. Typically, this allocation of units occurs either at the time of activation of the product 17 or at the time that product use is enabled on a particular platform (user CPU 16). The units typically remain allocated to the product 17 throughout the period that the product is running or is enabled to run.

Upon termination of processing or disabling, the allocated units are deallocated and made available for allocation to other instances of the software product 17 (other users 16 activating the product). In general, as long as any license units remain unallocated in field 42, the holder of the license is contractually authorized to increase his utilization of the licensed product. The usage does not deplete the license, however, as the units are returned to the units-available field 42 after a user is finished, and can be granted again to another user.

A consumptive unit based license, indicated in policy field 43, grants to the holder a specific number of initial license units (from field 42) and specifies the policy used to account for the consumption of those units. A software product 17 which is being managed by a consumptive license will cause an appropriate number of license units to be consumed to reflect the services provided by the product. Once consumed, units cannot be reused. Thus, the number of units available for future use declines upon every use of the licensed software product 17. This may also be referred to as a "metered" policy, being conceptually similar to measured consumption of electricity, water, etc. When the number of available units in field 42 reaches zero, the license may require that further use of the product is prohibited, or, the agreement may permit continued decrementing of the number of available units; the

result is the accumulation of a negative number of available units in the field 42. It is anticipated that most consumptive unit based licenses will consider negative units to represent an obligation of the license holder to pay the license issuer 25.

5 The transaction log 24 maintains an audit trail for providing a record of the units used in a consumptive license.

Referring to Figure 3, the major elements of the management policy are set forth in a table, where the possible entries for the fields 43, 44, 45 and 46 are listed. For the style entry

10 43, the possibilities are allocative and consumptive as just described, plus a category called "private" which represents a style of management undefined at present but instead to be created especially for a given product, using its own unique algorithm. It is expected that most licenses may be

15 administered using the named alternatives of Figure 3, but to allow for future expansion to include alternatives not presently envisioned, or to permit special circumstances for unique software, the "private" choices are included, which merely mean that the product 17 will generate its own conditions of

20 use. It is important to note that, except for the "private" alternative, the license management is totally in control of the license management program 11 on the license server 10 (or delegatee 13), rather than at the product 17. All the product 17 does, via the unit 18, is to make the request inquiry to the

25 server 10 via the client interface 31, and

report when finished.

The context field 44 specifies those components (sub-contexts) of the execution-context name which should be used in determining if unit allocations are required. License data is always used or allocated within, or for the benefit of, some named licensing context, and context can include "platform contexts" and "application contexts". Platform contexts are such things as a specific network, an execution domain, a login domain, a node, a process ID or a process family, a user name, a product name, an operating system, a specific hardware platform, as listed in Figure 3. Applications contexts are information supplied from the application (the product 17), such as may be used in a "private" method of determining license availability. The context name can use several of these, in which case the context name is constructed by concatenating the values of all subcontexts into a single context name, e.g., a VAX 3100 platform using VMS operating system.

The duration field 45 defines the duration of an allocation of license units to a specific context or the duration of the period which defines a valid consumptive use. For durations of type "Assignment," the specification of a reassignment constraint is also provided for, as discussed below. There are three types of duration, these being "transaction," "assignment" and "immediate" as seen in Figure 3.

The transaction duration type, when specified for an allocative policy, indicates that license units should be allocated to the specified context upon receipt of a license request and that those units should be deallocated and returned to the pool of available units upon receipt of a corresponding license release from a user node 16. Abnormal termination of the process or context having made the original license request will be semantically equivalent to a license release. On the other hand, when specified for a consumptive policy, this duration type indicates that license units should be allocated to the specified context upon receipt of a license request and permanently removed from the available units pool (field 42) upon receipt of a license release which reflects successful completion of the transaction. Upon receipt of a license release which carries an error status or upon abnormal termination of the processor context having made the original license request, the allocated units will be deallocated and returned to the pool of available units (field 42).

The assignment duration type in Figure 3 (field 45 of Figure 2) imposes the constraint that the required units must have been previously assigned to a specific context. The sub-contexts which must be specified in the assignment are those given in the context-template. A "reassignment constraint" may be imposed, and this is a limitation on how soon a reassignment can be made. For example, a reassignment constraint of 30-days

would require that units assigned to a specific context could not be reassigned more often than every 30-days; this would prevent skirting the intent of the license by merely reassigning units whenever a user of another context made a request allocation call for the product. Related to this assignment constraint, a "reallocation limit" may also be imposed, to state the minimum duration of an allocation; where there is a context template of process, the intent is to count the number of uses of the software product at a given time, but where software runs in batch rather than interactive mode it may run very quickly on a powerful machine, so a very few concurrent uses may permit almost unlimited usage - by imposing a reallocation constraint of some time period, this manner of skirting the intent of the license may be constrained.

The immediate duration type (field 45 of Figure 2) is used to indicate that the allocation or consumption of an appropriate number of license units from the pool of available units (field 42) should be performed immediately upon receipt of a license request. Receipt of license release or abnormal terminations will then have no impact on the license management system. When specified as the duration for an allocative policy, the effect will be simply to check if an appropriate number of license units are available at the time of a license request. When specified as the duration for a consumptive policy, the effect will be to deduct the appropriate number of license units

from the available pool at the time of a license request, and, thereafter, abnormal termination, such as a fault at the user CPU 16 or failure of the network link, will not reinstate the units.

5 The LURDM or license unit requirement determination method, field 46, has the alternatives seen in Figure 3 and stores information used in calculating the number of units that should be allocated or consumed in response to a license request. If this field specifies a table lookup kind, this
10 means license unit requirements are to be determined by lookup in the LURT (field 47) which is associated with the current license. If a constant kind is specified, this indicates that the license units requirements are constant for all contexts on which the licensed product or product feature may
15 run. A private LURDM specifies that the license unit requirements are to be determined by the licensed product 17, not by the license management facility 11. The license unit requirements tables (LURTs) provide a means by which issuers of licenses can store information describing the relation between
20 context (or row selector) and unit requirements.

The license units requirements determination method (LURDM) must specify "table lookup" for the LURT to be used, and if so a row selector must be specified, where a valid row selector is any subcontext, e.g., platform ID, user name, time of day, etc. An
25 example of an LURT fragment is shown in Figure 4,

illustrating the license unit requirements table mechanism. In this example, the row selector is "platform-ID" so the platform-ID value determines which row is used. The issuer of this LURT of Figure 4 has established three unit requirement

5 tiers for use in determining the unit requirements for that issuer's products. The reason for the tiers is not mandated by the license management system, but the issuer 25 (actually the user of the program 26) would probably be establishing

10 three pricing tiers, each reflecting a different perspective on the relative utility of different platforms in supporting the use of various classes of product 17. The first column in Figure 4, Column A, specifies the use requirements for a class of products whose utility is highly sensitive to the characteristics of the specific platform on which they are run.

15 This can be seen by observing that the unit requirements are different for every row in Column A. Products which use the second column (Column B) appear to have a utility which is more related to the class of platform on which they run. This is indicated by the fact that all the PC platforms share a single

20 value which is different from that assigned to the VAX platform. The final column (Column C) is for use with a class of products which is only supported on the VAX platform. Figure 4 is of course merely an example, and the actual LURT created by the license document generator 26 and stored in the license

25 database 23 (as field 47 of the product use authorization 35) can be of any content of this general format, as desired by the

license issuer.

Instead of always selecting the rows in LURT tables according to the platform ID of the execution platform, in order to handle the breadth of business practices that need to be supported by the license management facility, the LURT mechanism is extended by providing a "row selector" attribute in the LURT class structure. No default is provided although it is expected that the normal value for the row selector attribute will be "platform ID."

10 In the system of patent 4,937,863, a concept similar to that of the LURT of Figure 4 was provided, with rows selected by the platform ID and columns selected by some arbitrary means, typically according to product type. The system of this invention allows flexibility in the selection of both LURT row and column while continuing to provide backwards compatibility for licenses defined within the constraints of U.S. patent 15 4,937,863.

Some examples will illustrate potential uses for the row selector attribute. A customer may only want to pay for the use of a product during one or two months of the year; the product may be FORTRAN and the reason for this request may be that the company has a fairly stable set of FORTRAN subroutines that are given regular "annual maintenance" only during the months of May and June. To handle this customer's needs, the FORTRAN product would generate an application subcontext which would contain a value representing the month of the year. Then, a LURT table 25

would be defined with twelve rows, one for each month of the year. In some column, probably column A, a negative one (-1) would be placed in each month except for May and June. These two months would contain some positive number. The product use authorization would then have a LURDM field specifying a LURT for use to determine the units requirement, and would name this custom LURT table. The effect would be that the PUA could only be used during the months of May and June since negative one is interpreted by license managers to mean "use not authorized." This mechanism could also be used to do "time of day" charging. Perhaps charging fewer units per use at night than during the day. Also, if a subcontext was used that contained a year value, a type of license would be provided that varied in its unit requirements as time passed. For instance, it might start by costing 10-units per use in 1991 but then cost one unit less every year as time passed, eventually getting to the point where the unit requirement was zero.

Another example is font names. A specific customer may purchase a license giving it the right to concurrent use of 100-units of a large font collection; some of the fonts may cost more to use than others. For instance, Times Roman might cost 10-units per use while New Century Schoolbook costs 20-units per use.

The problem is, of course, making sure that charges are

properly made. The solution is to build a LURT table with a specified application subcontext as its row selector. A row is then created for each font in the collection and in Column A of the LURT, the number of units required to pay for use of the font would be specified. The print server would then specify the name of a font as the value of the application subcontext whenever it does an `lm_request_allocation` call. This will allow charges to be varied according to font name.

A further example is memory size. Some products are more or less valuable depending on the size of memory available to support them. A software vendor wishing to determine unit requirements based on memory size will be able to do so by building LURT tables with rows for each reasonable increment of memory (probably 1-megabyte increments). Their applications would then sense memory size (using some mechanism not part of the license management facility) and pass a rounded memory size value to the license manager in a private context.

Other examples are environment and operating system. Some products may be valued differently depending on whether they are being run in an interactive mode or in batch. This can be accomplished by building LURT rows for each of the standard platform subcontexts that specify environment. Regarding operating system, it has been considered desirable by many to have a single product use authorization permit the use of a

product on any number of operating systems, this conflicts with some vendors policies who do not want to have to create a single price for a product that applies to all operating systems. Thus, if an operating system independent license were offered for a C compiler, the price would be the same on MS-DOS, VMS, and/or UNIX. Clearly, it can be argued that the value of many products is, in part, dependent on the operating system that supports them. By using a row selector of operating system (one of the standard platform subcontexts), license designers could, in fact, require different numbers of units for each operating system. However, it might be more desirable to base the row selection on a private application subcontext that normally had the same value as the operating system subcontext. The reason for this is that the license designer might want to provide a default value for operating system names that were unknown at the time the LURT rows were defined. If this is the case, the product would contain a list of known operating systems and pass the subcontext value of "Unknown" when appropriate. The LURT row for "Unknown" would either contain a negative one (-1) to indicate that this operating system was unsupported or it would contain some default unit requirement.

Another example is variable pricing within a group. One of the problems with a "group" license is that there is only one unit requirements field on the PUA for a group. Thus, all members of the group share a single unit requirement.

However, in those cases where all members of the group can be appropriately licensed with a constant unit requirement yet it is desired to charge different amounts for the use of each group member, a LURT can be built that has rows defined for each group member. The row selector for such a group would be the standard platform subcontext "product name."

Many different types of license can be created using different combinations of contexts, duration and policy from the table of Figure 3. As examples, the following paragraphs show some traditional licensing styles which can be implemented using the appropriate values of the product use authorization fields 43-46.

A "system license" as it is traditionally designated is a license which allows unlimited use of a product on a single hardware system. The correct number of units must be allocated to the processor in advance and then an unlimited product use is available to users of the system. The product use authorization would have in the context field 44 a context template for a node name, the duration field would be "assignment" and the policy style field 43 would be "allocative".

A "concurrent use" license is one that limits the number of simultaneous uses of a licensed product. Concurrent use license units are only allocated when the product is being used and each

simultaneous user of the licensed product requires their own units. In this case the context template, field 44, is a process ID, the duration field is "transaction" and the policy style 43 is "allocative".

5 A "personal use" license is one that limits the number of
named users of a licensed product. This style of licensing
guarantees the members of a list of users access to a product.
Associated with a personal use type of product use
authorization there is a list of registered users. The
10 administrator is able to assign these users as required up to
the limit imposed by the product use authorization;
the number of units assigned to each user is indicated by the
LURDM. It may be a constant or it may vary as specified in a
LURT. The context template is "user name", the duration is
15 "assignment", and the policy is "allocative".

A "site license" is one that limits the use of a licensed
product to a physical site. Here the product use authorization
contains for the context template either "network name" or
"domain name", the duration is "assignment" and the policy
20 style field 43 is "allocative".

Generally, a license to use a software product is priced
according to how much benefit can be gained from using the
product, which is related to the capacity of the machine it will

run on. A license for unlimited use on a large platform such as a mainframe, where there could be thousands of potential users at terminals, would be priced at a high level. Here the style would be "allocative", the context template = "node", the duration = "assignment" and the LURDM may be "Column A" - the units, however, would be large, e.g., 1000. At the other end of the scale would be a license for use on a single personal computer, where the field values would be the same as for the mainframe except the units would be "1". If a customer wanted to make the product available on the mainframe but yet limit the cost, he could perhaps get a license that would allow only five users at any given time to use the product; here the fields in the product use authorization would be: units = 5; style = allocative; context template = process; duration = transaction; LURDM = constant, 1-unit. This would still be priced fairly high since a large number of users may actually use the product if a session of use was short. A lower price would probably be available for a personal use license where only five named persons could use the product, these being identified only in the license server 10, not named by the license issuer 25. Here the fields in the product use authorization are: units = 5; style = allocative; context template = user name; duration = transaction; LURDM = constant, 1-unit.

An additional feature that may be provided for in the product use authorization 35 is license combination. Where

there are multiple authorizations for a product, license checking requests sent by user nodes 16 may be satisfied by combining units from multiple authorizations. Individual product use authorizations may prohibit combined use. Thus, a licensee may have a license to use a product 17 on an allocative basis for a certain number of units and on a consumptive basis for another number of units (this may be attractive from pricing standpoint); there might not be enough units available for a particular context from one of these licenses, so some units may be "borrowed" from the other license (product use authorization), in which case a combination is made.

The interface between the program executing on the client or user 16 and the license server 10 or its delegates 13 includes basically three procedure calls: a request allocation, a release allocation and a query allocation. Figure 5 illustrates in flow chart form some of the events occurring in this client interface.

The request allocation is the basic license checking function, a procedure call invoked when a software product 17 is being instantiated, functioning to request an allocation of license units, with the return being a grant or refusal to grant.

Note that a product may use request allocation calls at a number of points in executing a program, rather than only upon start-up; for example, a request allocation may be sent when making use of some particular feature such a special

graphics package or the like. The release allocation call is invoked when the user no longer needs the allocation, e.g., the task is finished, and this return is often merely an acknowledge; if the style is consumptive, the caller has the opportunity via the release allocation call to influence the number of units consumed, e.g.,

decrease the number due to some event. The query allocation call is invoked by the user to obtain information about an existing allocation, or to obtain a calling card, as will be described.

The request allocation, referred to as `lm_request_allocation()`, is a request that license units be allocated to the current context. This function returns a grant or denial status that can be used by the application programmer to decide whether to permit use of the product or product feature. The status is based on the existence of an appropriate product use authorization and any license management policies which may be associated with that product use authorization. License units will be allocated or consumed, if available, according to the policy statement found on the appropriate product use authorization. The product would normally call this function before use of a licensed product or product feature. The function will not cause the product's execution to be terminated should the request fail. The decision of what to do in case of failure to obtain allocation

of license units is up to the programmer. The arguments in a request allocation call are the product name, producer name, version, release date, and request extension. The product name, producer name, version and release date are the name of the software product, name of producer, version number and release date for specifically identifying the product which the user is requesting an allocation be made. The request extension argument is an object describing extended attributes of the request, such as units required, LURT column, private context, and comment. The results sent back to the calling node are a return code, indicating whether the function succeeded and, if not, why not, and a grant handle, returned if the function completes successfully, giving an identifying handle for this grant so it can be referred to in a subsequent release allocation call or query allocation call, for example.

The release allocation, referred to as `lm_release_allocation()`, is an indication from a user to the license manager to release or consume units previously allocated. This function releases an allocation grant made in response to a prior call to request allocation. Upon release, the license management style 38 determines whether the units should be returned to the pool of available units or consumed. If the caller had specified a request extension on the earlier call to request allocation which contained a units-required-attribute, and the number of units requested at

that time are not the number of units that should be consumed for the completed operation, the caller should state with the units-consumed argument how many units should be consumed. The arguments of the release allocation are: grant handle, units
5 consumed, and comment. The grant handle identifies the allocation grant created by a previous call to request allocation. The units-consumed argument identifies the number of units which should be consumed if the license policy is consumptive; this argument should only be used in combination
10 with an earlier call to request allocation which specified a units requirement in a request extension. Omission of this argument indicates that the number of units to be consumed is the same as the number allocated previously.

The comment argument is a comment which will be written to the
15 log file 24 if release units are from a consumptive style license or if logging is enabled. The result is a return code indicating if the function succeeded, and, if not, why not.

The query allocation, or `lm_query_allocation()`, is used by licensed products which have received allocations by a previous
20 request allocation call. The query is to obtain information from the server 10 or delegatee server 13 about the nature of the grant that has been made to the user and the license data used in making the grant, or to obtain a calling card (i.e., a request that a calling card be issued).

25 Typically, the item read by this query function is the token

field 52 which contains arbitrary information encoded by the license issuer and which may be interpreted as required by the stub 19 for the licensed product software 17, usually when a "private" allocation style or context is being employed.

5 The arguments in this procedure call are the grant handle, and the subject. The grant handle identifies the allocation grant created by a previous call to request allocation. The subject argument is either "product use authorization" or "calling card request"; if the former then the result will contain a public copy of the product use authorization. If this
10 argument is a calling card request and a calling card which matches the previous constraints specified in that request can be made available, the result will contain a calling card. If the subject argument is omitted, the result will contain an
15 instance of the allocation class. The results of the query allocation call are (1) a return code, indicating whether the function succeeded, and, if not, why not, and (2) a result, which is either an allocation, a product use authorization or a calling card, depending on type and presence of the subject
20 argument.

Referring to Figure 5, the flow chart shows the actions at the client in its interface with the server. When the software product 17 is to be invoked, the unit 18 is first executed as indicated by the block 60, and the first action is to make a
25 request allocation call via the stub 19, indicated by the block

61. The client waits for a return, indicated by the loop 62, and when a return is received it is checked to see if it is a grant, at decision block 63. If not, the error code in the return is checked at block 64, and if a return code indicates a
5 retry is possible, block 65, control passes back to the beginning, but if no retry is to be made then execution is terminated. If the policy is to allow use of the product 17 without a license grant, this function is separately accounted for. If the decision point 63 indicates a grant was made, the
10 grant handle is stored, block 66, for later reference. The program 17 is then entered for the main activities intended by the user. During this execution of product 17, or before or after, a query allocation call can be made, block 67, though this is optional and in most cases not needed. When execution
15 of the program 17 is completed, the grant handle is retrieved, block 68, and a release allocation call is made, block 69. A loop 70 indicates waiting for the return from the server, and when the return received it is checked for an error code as before, and a retry may be appropriate. If the release is
20 successfully acknowledged, the program exits.

Referring to Figure 6, the actions of the server 10 or delegatee server 13 in executing the license management program 11 or 14, for the client interface, are illustrated in flow diagram form. A loop is shown where the server program
25 is checking for receipt of a request, release or query call from

its clients. The call would be a remote procedure call as discussed above, and would be a message communicated by a network, for example. This loop shows the decision blocks 71, 72 and 73. If a release allocation call is received, a list of products for which authorizations are stored is scanned, block 5 74, and compared to the product identity given in the argument of the received call, block 75. If there is no match, an error code is returned to the client, block 76, and control goes back to the initial loop. If the product is found, the authorization is retrieved from the database 23, block 77 (there may be more 10 than one authorization for a given product, in which case all would be retrieved, but only one will be referred to here) and all of the information is matched and the calculations made depending upon the management policy of Figures 3 and 4, 15 indicated by the decision block 78. If a grant can be made, it is returned as indicated at block 79, or if not an error code is returned, block 80. If a release allocation call is received, indicated by a positive at the decision block 72, the grant handle in the argument is checked for validity at block 81. If 20 no match is found, an error code is returned, block 82, and control passes back to the initial loop. If the handle is valid, the authorization for this product is retrieved from the database 23 at block 83, and updated as indicated by the block 84. For example, if the license management style is allocative, 25 the units are returned to the available pool. Or, in some cases, no update is needed. The authorization is stored again

in the database, block 85, and a return made to the client, block 86, before control passes back to the initial loop. If the decision block 73 indicates that a query allocation call is received, again the grant handle is checked at block 87, and an error code returned at block 88 if not valid. If the grant handle matches, the authorization is retrieved from the database 23, at block 89, and a return is made to the client giving the requested information in the argument, block 90.

The basic allocation algorithm used in the embodiment of the license management system herein described, and implemented in the method of Figures 5 and 6, is very simple and can handle a very large proportion of known license unit allocation problems. However, it should be recognized that a more elaborate and expanded algorithm could be incorporated. Additions could be made in efforts to extend the allocation algorithm so that it would have specific support for optimizing unit allocation in a wider variety of situations. Particularly, sources of non-optimal allocations occurring when using the basic allocation algorithm are those that arise from combination and reservation handling.

The first step is formation of full context. The client stub 19 is responsible for collecting all specified platform and application subcontexts from the execution environment of the product 17 and forwarding these collected subcontexts to the license management server 13 or 10. The collection of

subcontexts is referred to as the "full context" for a particular license unit allocation request.

The next step is retrieval of the context template. When the license manager receives an `lm_request_allocation`, it will
5 look in its list of available product use authorizations (PUA) to determine if any of them conform to the product identifier provided in the `lm_request_allocation` call. The product identifier is composed of: product name, producer, version, release date. If any match is found, the license manager will
10 extract from the matching PUA the context template. This template is composed of a list of subcontexts that are relevant to the process of determining unit requirements. Thus, a context template may indicate that the node-ID subcontext of a specific full context is of interest for the purposes of unit
15 allocation. The context template would not specify any specific value for the node-ID; rather, it simply says that node-ID should be used in making the allocation computation.

The next step is masking the full context. Having retrieved the context template, the license manager will then
20 construct an "allocation context" by filtering the full context to remove all subcontexts which are not listed in the context template. This allocation context is the context to be used in determining allocation requirements.

Then follows the step of determining if the request is new. The license manager maintains for each product use authorization a dynamic table which includes the allocation contexts of all outstanding allocations for that PUA (i.e., allocations that have been granted but have not yet been released). Associated with each entry in this table is some bookkeeping information which records the number of units allocated, the full context, etc. To determine if a recent `lm_request_allocation` requires an allocation of units to be made, the license manager compares the new allocation context with all those allocation contexts in the table of outstanding allocations and determines if an allocation has already been made to the allocation context. If the new allocation context does not already exist in the table, an attempt will be made to allocate the appropriate number of units depending on the values contained in the LURDM structure of the PUA and any LURTs that might be required. If an allocation context similar to that specified in the new allocation request does exist in the table, the license manager will verify that the number of units previously allocated are equal to or greater than the number of units which would need to be allocated to satisfy the new allocation request. If so, the license manager will return a grant handle to the application which indicates that the allocation has been made (i.e., it is a "shared allocation" - the allocated units are shared between two requests.) If not, the license manager will attempt to allocate a number of units

equal to the difference between the number previously allocated and the number of units required.

The step of releasing allocations (Fig. 6, blocks 84-85) occurs when the license manager receives an
5 lm_release_allocation call; it will remove the record in its dynamic allocation table that corresponds to the allocation to be released.

Having done this, the license manager will then determine if the allocation to be removed is being shared by any other allocation
10 context. If so, the units associated with the allocation being released will not be released. They will remain allocated to the remaining allocation contexts. Some of the units might be released if the license manager determines that the number of allocated units exceeds the number needed to satisfy the
15 outstanding allocation contexts. If this is the case, the license manager will "trim" the number of allocated units to an appropriate level.

In summary, the two things that make this algorithm work are (1) the basic rule that no more than one allocation will be
20 made to any single allocation context, and (2) the use of the context template to make otherwise dissimilar full contexts appear to be similar for the purposes of allocation.

The license designer's task, when defining basic policy, is

then to determine which contexts should appear to be the same to the license manager. If the license designer decides that all contexts on a single node should look the same (context template = node-ID), then any requests that come from that node will
5 all share allocations. On the other hand, a decision that all contexts should be unique (i.e., context template = process-ID) will mean that allocations are never shared.

This mechanism permits the system of the invention to dispose of the cumbersome, explicit support of license types
10 having different scope such as the cluster licenses, node licenses, and process licenses found in prior license management systems including that of patent 4,937,863. Instead of defining a limited set of scopes (cluster, node, etc.), the system of this invention provides a general mechanism which
15 allows an effectively unlimited range of allocation scopes to be defined.

Transitive licensing, as referred to above, is supported by the system of the invention by (1) calling authorizations, which are statements made in field 49 of the product use authorization
20 35 for one product (the "caller") to permit that product to call another product (the "callee"), and, (2) caller authorizations, which are statements made in field 49 of the product use authorization for one product (the "callee") to permit it to be called by another product (the

"caller").

If calling or caller authorizations are to be exploited by products, then whenever one product calls another product, it must pass the callee a calling card 49a. This calling card 49a is an encoding of an identification of the caller as well as a statement by the license management system that a license unit allocation has been made to the caller which is passing the calling card. This calling card is then passed by the callee to the license management system for validation and, if the either the product use authorization of the caller carries an appropriate calling authorization or the product use authorization of the callee carries an appropriate caller authorization, the use of the callee by the caller will be authorized without requiring any additional license unit allocations.

Referring to Figure 7, the intercomponent interactions that occur when either calling or caller authorizations are being used are illustrated. This figure shows a license management server 10, a caller product 17a named "Product-1" and a callee product 17b named "Product-2". When Product-1 starts to run, it will make an `lm_request_allocation` call to the license management server 10 to obtain a grant handle for an allocation of some number of units of the Product-1 license. Either immediately, or at some later time, but always prior to

making a call to Product-2, Product-1 will call
lm_query_allocation, passing the grant handle received earlier
and specifying that it wants a calling card for the product
named "Product-2." If the field 49 of the product use
5 authorization 35 used to satisfy the grant represented by the
grant handle carries a calling authorization in field 49
naming "Product-2," the license manager will create a calling
card 49a which includes the statement that a calling
authorization exists and pass this calling card back to
10 Product-1. If the calling authorization does not exist, the
calling card passed to Product-1 will contain a statement to
that effect.

Once Product-1 has successfully obtained a calling card 49a
from the license manager, it will then make a call to Product-2,
15 passing the calling card along with any other initialization
parameters that would normally be used when starting Product-2.
Product-2 will then pass that calling card to the license
manager as part of its lm_request_allocation call and the
license manager will determine if the calling card is valid.
20 Note that calling cards become invalid once
the process which received the calling card makes an
lm_release_allocation call or terminates abnormally. If the
calling card is valid, and it indicates that a calling
authorization is present, the license manager will verify this
25 statement and if found to be true, will return a grant handle to

Product-2. If, on the other hand, the calling card carries an indication that no calling authorization is present, the license manager will attempt to find a product use authorization for Product-2 that contains a caller authorization naming Product-1 as an authorized caller. If the caller authorization is found, a grant handle will be passed back to Product-2. If not, the license manager will ignore the calling card and proceed with the normal `lm_request_allocation` logic.

The requirement to be passing calling cards between products requires that both the caller and the callee be "aware" of the fact that calling and caller authorizations may be used. This is one of the few examples of a requirement for a product 17 to become actively involved in the licensing problem when using the licensing management system of the invention. However, since the use of calling/caller authorizations is a fairly "sophisticated" and powerful feature, it is considered acceptable to impose this burden on application coders.

While this invention has been described with reference to specific embodiments, this description is not meant to be construed in a limiting sense. Various modifications of the disclosed embodiments, as well as other embodiments of the invention, will be apparent to persons skilled in the art upon reference to this description. It is therefore contemplated

NOT FURNISHED UPON FILING

WHAT IS CLAIMED IS:

1 1. A method (11) of managing use of licensed software
2 items, comprising the steps of:

3 maintaining a store (23) of license authorizations for said
4 software items; each license authorization including an
5 indication of license management policy (43) (14) for a
6 software item having plurality of sets of policy components,
7 said policy components in each set providing alternatives, to
8 produce a composite policy simultaneously using a policy
9 component from each of said sets;

10 sending a request by a user (16) of one of said software
11 items to obtain permission to use said software item; said
12 request identifying the user and said software item (38, 36);

13 accessing said store to obtain information from said
14 license authorization for said software item, in response to
15 said request, and comparing (75) said identification of said
16 user and said software item with said information, and with
17 constraints imposed by said plurality of sets of license
18 management policy components, to produce a grant (79) or refusal
19 of said request;

20 sending said grant or refusal to said user.

1 2. A method according to claim 1 wherein said store is
2 maintained by a license server (10), and said request is sent to
3 said server.

1 3. A method according to claim 1 wherein each one of said
2 sets of license management policy components includes a
3 plurality of specified ways of allotting
4 units of license (26) use to said user.

1 4. A method according to claim 3 wherein one of said sets
2 (43) of license management policy items includes an allocative
3 policy and a consumptive policy, where said allocative policy
4 permits reuse of units allocated to a user after the
5 user completes a requested use, and where said consumptive
6 policy does not permit said reuse.

1 5. A method according to claim 2 wherein said sets include
2 a set of contexts, where a context identifies an environment,
3 location of use or condition of use of a licensed product
4 wherein said context includes at least one of two types of
5 contexts including platform contexts and application contexts.

1 6. A method according to claim 5 wherein said platform
2 context can include a network, an execution domain, a node, a
3 user name, an operating system, a platform ID, and a CPU type
4 wherein said application context can include information from
5 said software item.

1 7. A method according to claim 1 wherein said request is

2 in the form of a remote procedure call, and said grant or
3 refusal sent to said user is a return of said procedure call
4 wherein said license document is a data arrangement specified as
5 a product use authorization, and said product use authorization
6 is received by said server from an issuer.

1 8. A method according to claim 1 wherein there are at
2 least three of said sets of alternative policy components, and
3 wherein said sets include a style component providing
4 alternative ways of allocating license units,
5 a context component identifying an environment, location of
6 use or condition of use of a licensed product; and
7 a usage requirements component providing alternative ways
8 of determining the usage requirement for said users.

1 9. A method according to claim 8 further including a unit
2 requirements table stored with said license document and
3 providing a plurality of alternative license unit values for
4 different users, including the step of selecting a row of said
5 table based upon said context component.

1 10. A method according to claim 1 including the step of
2 delegating a part of a license granted in said license document
3 to another license manager, and sending said requests from
4 additional users to said another manager, a method according to
5 claim 15 wherein said manager, said another manager, said users,

6 and said additional users, are all nodes on at least one computer
7 network.

1 11. A method according to claim 1 including the steps of:
2 sending a request by said user to said server to obtain
3 permission to make a call to another program at a second user;
4 accessing said store to obtain information from said
5 license document for said program giving a calling
6 authorization, in response to said request, and producing a
7 calling card if said calling authorization is in said
8 information;
9 sending said calling card to said user from said server;
10 sending said calling card by said user to said second user
11 in making a call to said second user;
12 sending a request by said second user to said server to
13 verify said calling card, and receiving a return from said
14 server; and
15 completing said call from said user to said second user.

1 12. A system (11) for managing use of licensed software
2 products, comprising:
3 means for maintaining a store (23) of license documents,
4 one for each said program; each license document including an
5 indication of license policy having plurality of sets of
6 alternative policy components;

7 means for sending a request from a user of one of said
8 programs to obtain permission to use said program; said request
9 identifying the user and said program;

10 means for accessing said store to obtain information from
11 said license document for said program, in response to said
12 request, and for comparing said identification of said user and
13 said program with said information, and with constraints imposed
14 by said sets of policy components, to produce a grant or
15 refusal of said request;

16 means for sending said grant or refusal to said user.

1 13. A system according to claim 12 wherein said means for
2 maintaining, said means for accessing and said means for sending
3 to said user are all located at a server on a distributed, and
4 said means for sending a request is located at a user node on
5 said network, and wherein said license policy includes an
6 allocative policy and a consumptive policy, where said
7 allocative policy permits reuse of units allocated to a user
8 after the user completes a requested use, and where said
9 consumptive policy does not permit said reuse.

1 14. A system according to claim 13 wherein said sets
2 include a set of contexts, where a context identifies an
3 environment, location of use or condition
4 of use of a licensed product, wherein said context includes at
5 least one of two types of contexts including platform contexts

NOT FURNISHED UPON FILING

1 18. A system according to claim 13 including:

2 means for sending a request by said user to said server to
3 obtain permission to make a call to another program at a second
4 user;

5 means for accessing said store to obtain information from
6 said license document for said program giving a calling
7 authorization, in response to said request, and for producing a
8 calling card if said calling authorization is in said
9 information;

10 means for sending said calling card to said user from said
11 server;

12 means for sending said calling card by said user to said
13 second user in making a call to said second user;

14 means for sending a request by said second user to said
15 server to verify said calling card, and receiving a return from
16 said server, before completing said call from said user to said
17 second user, and wherein said requests are in the form

18 of remote procedure calls on a network, and said grant or
19 refusal and said calling card are sent to said user by a return
20 of a remote procedure call on said network.

1 19. A system according to claim 12 including means for
2 delegating a part of a license granted in said license document
3 to another license manager, and for sending said requests from
4 additional users to said another manager.

1 20. A method of operating a distributed network,
2 comprising the steps of:

3 maintaining at a server on said network a store of program
4 use authorizations;

5 sending a request by a first user on said network to said
6 server to obtain permission to make a call to a program at a
7 second user on said network;

8 accessing said store to obtain information from said
9 authorization for a program, in response to said request, to
10 determine if said call is authorized and to produce a calling
11 card if said call is authorized;

12 sending said calling card to said to said first user;

13 sending said calling card by said first user to said second
14 user in making a call to said second user;

15 sending a request by said second user to said server to
16 verify said calling card, and receiving a return from said
17 server; and completing the operation requested by said call from
18 said first user to said second user and making a return.

1 21. A method of managing use of licensed software,
2 comprising the steps of:

3 generating at a license issuer a license authorization for
4 each product of said licensed software, and sending said license
5 authorization to a license manager;

6 maintaining at a license manager a store of said license

7 authorizations; each license authorization including an
8 indication of license management policy for a product, said
9 indication having multiple components; said license manager
10 being functionally separate from said license issuer and being
11 unable to perform said step of generating;

12 sending to said license manager a request by a user of one
13 of said products to obtain permission to use said product; said
14 request identifying the user and said product; said user being
15 functionally separate from said license manager and said
16 license issuer;

17 accessing said store by said license manager to obtain
18 information from said license authorization for said product, in
19 response to said request, and comparing said identification of
20 said user and said product with said information,
21 and with constraints imposed by said components of said license
22 management policy, to produce a grant or refusal of said
23 request;

24 sending said grant or refusal from said license manager to
25 said user.

1 22. A method of managing use of licensed software
2 products, comprising the steps of:

3 maintaining at a license manager a store of license
4 authorizations; each license authorization including an
5 indication of license management policy for a product, said
6 indication having multiple components, including a start time

7 and an end time for a license period;

8 sending to said license manager a request by a user of one
9 of said products to obtain permission to use said product; said
10 request identifying the user and said product;

11 accessing said store by said license manager to obtain
12 information from said license authorization for said product, in
13 response to said request, and comparing said identification of
14 said user and said product with said information, and with
15 constraints imposed by said components of said license
16 management policy and the time of said request compared to said
17 start time and end time, to produce a grant or refusal of said
18 request;

19 sending said grant or refusal from said license manager to
20 said user.

AMENDED CLAIMS

[received by the International Bureau on 23 September 1992 (23.09.92);
original claims 1,2 and 7-10 amended; remaining claims unchanged (2 pages)]

1 1. A method (11) of managing use of licensed software items,
2 comprising the steps of:
3 maintaining a store (23) of license authorizations for
4 said software items; each license authorization including an
5 indication of license management policy (43) (14) for a
6 software item having plurality of sets of policy components,
7 said policy components in each set providing alternatives, to
8 produce a composite policy simultaneously using a policy
9 component from each of said sets;
10 sending a request by a user (16) of one of said software
11 items to obtain permission to use said software item; said
12 request identifying the user and said software item (38, 36);
13 accessing said store to obtain information from said
14 license authorization for said software item, in response to
15 said request, and comparing (75) said identification of said
16 user and said software item with said information, and with
17 constraints imposed by said plurality of sets of license
18 management policy components, to produce a grant (79) or
19 refusal of said request;
20 sending said grant or refusal to said user.

1 2. A method according to claim 1 wherein said store is
2 maintained by a license server (10), and said request is sent
3 to said server.

2 7. A method according to claim 1 wherein said request
3 is in the form of a remote procedure call, and said grant or
4 refusal sent to said user is a return of
5 said procedure call wherein said license document is a data
6 arrangement specified as a product use authorization, and said
7 product use authorization is received by said server from an
8 issuer.

1 8. A method according to claim 1 wherein there are at
2 least three of said sets of alternative policy components, and
3 wherein said sets include a style component providing
4 alternative ways of allocating license units,
5 a context component identifying an environment,
6 location of use or condition of use of a licensed product; and
7 a usage requirements component providing
8 alternative ways of determining the usage requirement for said
9 users.

1 9. A method according to claim 8 further including a
2 unit requirements table stored with said license document and
3 providing a plurality of alternative license unit values for
4 different users, including the step of selecting a row
5 of said table based upon said context component.

1 10. A method according to claim 1 including the step of
2 delegating a part of a license granted in said license
3 document to another license manager, and sending said requests
4 from additional users to said another manager, wherein said
5 manager, said another

INTERNATIONAL SEARCH REPORT

PCT/US 92/03608

International Application No

I. CLASSIFICATION OF SUBJECT MATTER (If several classification symbols apply, indicate all)⁶

According to International Patent Classification (IPC) or to both National Classification and IPC
Int.Cl. 5 G06F1/00

II. FIELDS SEARCHED

Minimum Documentation Searched⁷

Classification System	Classification Symbols
Int.Cl. 5	G06F

Documentation Searched other than Minimum Documentation
to the Extent that such Documents are Included in the Fields Searched⁸

III. DOCUMENTS CONSIDERED TO BE RELEVANT⁹

Category ¹⁰	Citation of Document, ¹¹ with indication, where appropriate, of the relevant passages ¹²	Relevant to Claim No. ¹³
Y	EP,A,0 332 304 (DIGITAL EQUIPMENT CORPORATION) 13 September 1989 cited in the application	1-3,5, 8-12, 18-22
A	see figure 1 see column 3, line 31 - column 7, line 55 ---	4,6,7,13
Y	EP,A,0 398 492 (IBM) 22 November 1990 see figures 1,4A,4B,5,6 see page 5, line 45 - page 6, line 54 see page 12, line 4 - line 40 ---	1-3,5, 8-12, 18-22

¹⁰ Special categories of cited documents:^{"A"} document defining the general state of the art which is not considered to be of particular relevance^{"E"} earlier document but published on or after the international filing date^{"L"} document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)^{"O"} document referring to an oral disclosure, use, exhibition or other means^{"P"} document published prior to the international filing date but later than the priority date claimed^{"T"} later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention^{"X"} document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step^{"Y"} document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.^{"&"} document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search

10 SEPTEMBER 1992

Date of Mailing of this International Search Report

17. 09. 92

International Searching Authority

EUROPEAN PATENT OFFICE

Signature of Authorized Officer

WEISS P.

**ANNEX TO THE INTERNATIONAL SEARCH REPORT
ON INTERNATIONAL PATENT APPLICATION NO. US 9203608
SA 61767**

This annex lists the patent family members relating to the patent documents cited in the above-mentioned international search report.
The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information. 10/09/92

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
EP-A-0332304	13-09-89	US-A-	4937863	26-06-90
		JP-A-	2014321	18-01-90
EP-A-0398492	22-11-90	CA-A-	2016224	15-11-90
		JP-A-	3005868	11-01-91